1. Kudu产生背景介绍

在 kudu 出现之前，主要有两种数据存储方式：

静态数据：

> 以 HDFS 引擎作为存储引擎，适用于高吞吐量的离线大数据分析场景。 这类存储的局限性是数据无法进行随机的读写。

动态数据：

> 以 HBase、Cassandra 作为存储引擎，适用于大数据随机读写场景。局限性是批量读取吞吐量远不如 HDFS，不适用于批量数据分析的场景。

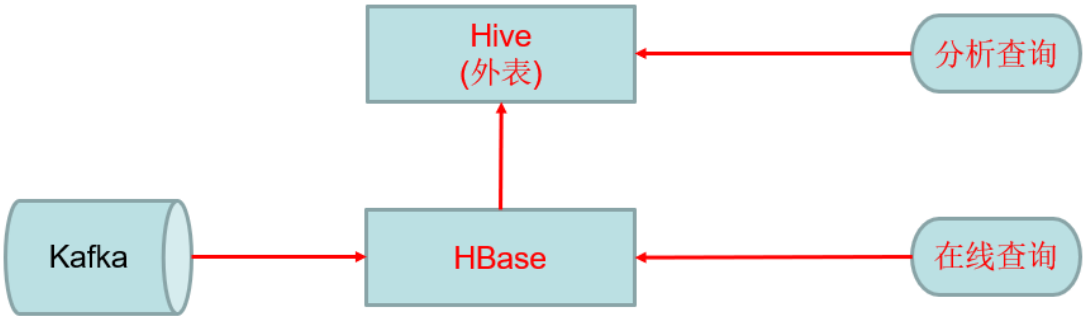
数据形态	存储	场景	局限性
静态数据	HDFS(Parquet) + Hive/Impala	适用于高吞吐量的离线大数据分析场景	这类存储的局限性是数据无法进行随机的读写
动态数据	以 HBase/Cassandra 作为存储引擎	适用于大数据随机读写场景	不适合高吞吐的大数据离线分析

从上面分析可知，这两种数据在存储方式上完全不同，进而导致使用场景完全不同，但在真实的场景中，边界可能没有那么清晰，面对既需要随机读写，又需要批量分析的大数据场景，该如何选择呢？

最终导致几个问题：

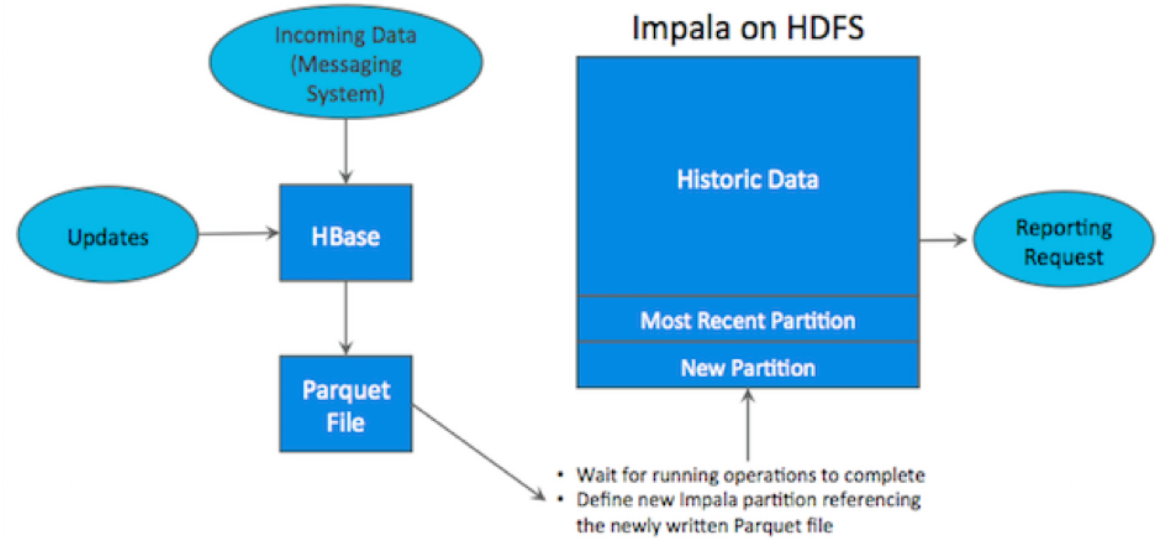
- 数据过度冗余：数据需要存储多份以支撑多个应用，这样造成了存储等资源的浪费。
- 架构复杂导致开发、运维、测试成本高：同时维护多套存储系统，架构复杂，开发、运维、测试成本相对较高。
- 数据不一致容易误解：多套数据由于程序 bug 或者其他原因很容易出现数据不一致的情况，往往会造成业务方的误解。

为了解决上述问题业界做了很多尝试，例如 HBase + Hive 整合：



上述方案虽然在一定程度上起到了作用，但是依然改变不了 HBase 不适合高吞吐量离线大数据分析的事实。

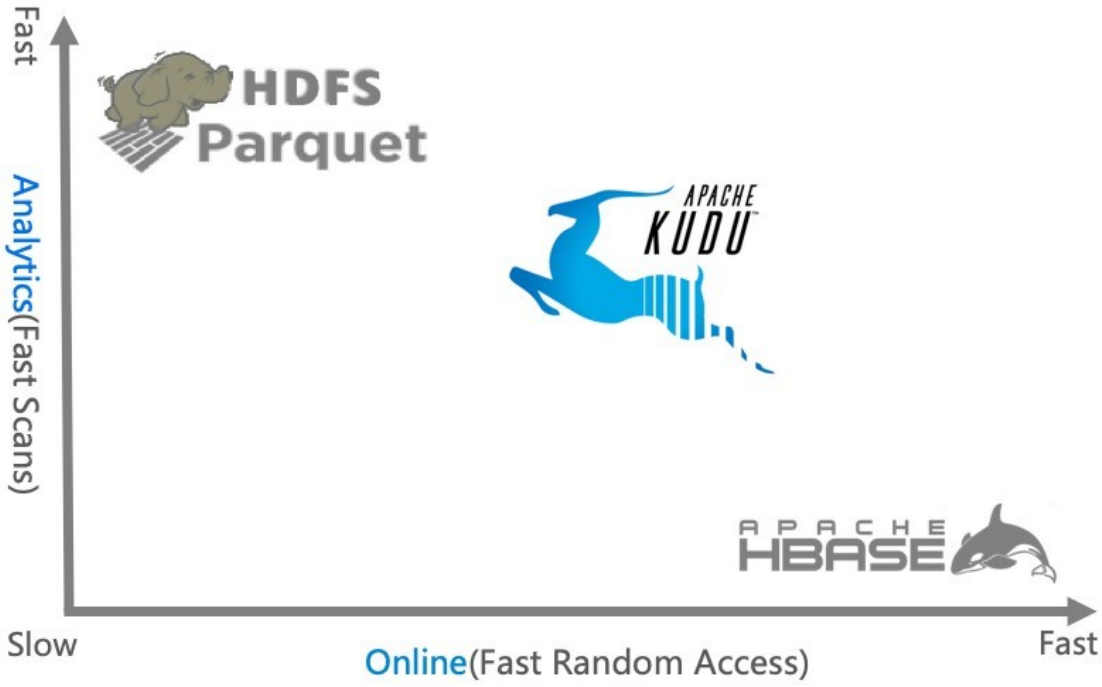
这个场景中，单种存储引擎无法满足业务需求，我们需要通过多种大数据工具组合来满足这一需求，如下图所示：



如上图所示，数据实时写入 HBase，实时的数据更新也在 HBase 完成，为了应对 OLAP 需求，我们定时将 HBase 数据写成静态的文件（如：Parquet）导入到 OLAP 引擎（如：Impala、Hive）。这一架构能满足既需要随机读写，又可以支持 OLAP 分析的场景，但他有如下缺点：

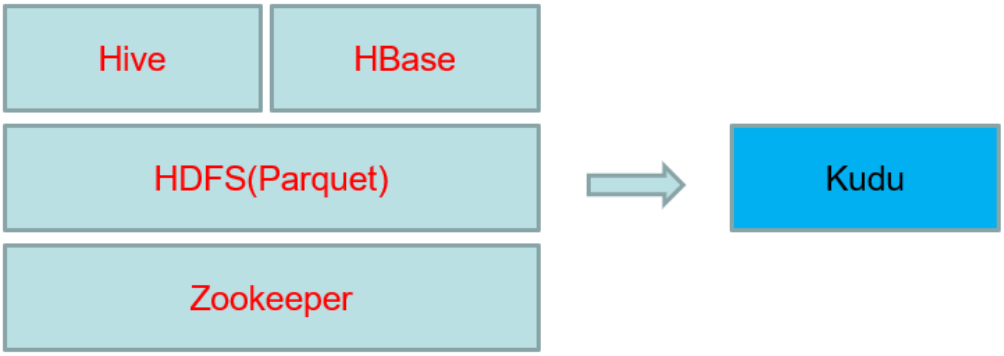
> 1、架构复杂。从架构上看，数据在 HBase、消息队列、HDFS 间流转，涉及环节太多，运维成本很高。并且每个环节需要保证高可用，都需要维护多个副本， 存储空间也有一定的浪费。最后数据在多个系统上，对数据安全策略、监控等都提出了挑战。
>
> 2、时效性低。数据从 HBase 导出成静态文件是周期性的，一般这个周期是一天（或一小时），在时效性上不是很高。
>
> 3、难以应对后续的更新。真实场景中，总会有数据是延迟到达的。如果这些数据之前已经从 HBase 导出到 HDFS，新到的变更数据就难以处理了，一个方案是把原有数据应用上新的变更后重写一遍，但这代价又很高。

为了解决上述架构的这些问题， Kudu 应运而生。Kudu 的定位是 Fast Analytics on Fast Data，**是一个既支持随机读写、又支持 OLAP 分析的大数据存储引擎。**



从上图可以看出，Kudu 是一个折中的产品，在 HDFS 和 HBase 这两个偏科生中平衡了随机读写和批量分析的性能。从 Kudu 的诞生可以说明一个观点：底层的技术发展很多时候都是上层的业务推动的，脱离业务的技术很可能是空中楼阁。

当然 Kudu 身上还有很多概念或者标签，有分布式文件系统（好比 HDFS），有一致性算法（好比 Zookeeper），有 Table（好比 Hive Table），有 Tablet（好比 Hive Table Partition），有列式存储（好比 Parquet），有顺序和随机读取（好比 HBase），所以看起来 Kudu 是一个轻量级的 **HDFS** + Zookeeper + Hive + Parquet + **HBase**，除此之外，Kudu 还有自己的特点，快速写入+读取，使得 kudu + Impala 非常适合 OLAP 场景，尤其是 Time-series 场景。



HDFS 与 HBase 数据存储的缺点。目前数据存储有了 HDFS 与 HBase，为什么还要额外的弄一个 kudu 呢?

> HDFS：使用列式存储格式Apache Parquet，Apache ORC，适合离线分析，不支持单条纪录级别的update操作，随机读写性能差
>
> HBASE：可以进行高效随机读写，却并不适用于基于SQL的数据分析方向，大批量数据获取时的性能较差。

正因为 HDFS 与 HBASE 有上面这些缺点，Kudu 较好的解决了 HDFS 与 HBASE 的这些缺点，它不及 HDFS 批处理快，也不及 HBase 随机读写能力强，但是反过来它比 HBase 批处理快（适用于 OLAP 的分析场景），而且比 HDFS 随机读写能力强（适用于实时写入或者更新的场景），这就是它能解决的问题。

2. Kudu 整体介绍

Kudu 是 Cloudera 开源的运行在 Hadoop 平台上的列式存储系统，拥有 Hadoop 生态系统应用的常见技术特性，运行在一般的商用硬件上，支持水平扩展、高可用。Apache Kudu 是由 Cloudera 开源的存储引擎，可以同时提供低延迟的随机读写和高效的数据分析能力。它是一个融合 HDFS 和 HBase 的功能的新组件，具备介于两者之间的新存储组件。Kudu 支持水平扩展，并且与 Cloudera Impala 和 Apache Spark 等当前流行的大数据查询和分析工具结合紧密。

Kudu 是一个分布式列式存储引擎/系统，由 Cloudera 开源后捐献给 Apache 基金会很快成为顶级项目。 用于对大规模数据快速读写的同时进行快速分析。

Kudu 是 Cloudera 开源的运行在 Hadoop 平台上的列式存储系统，拥有 Hadoop 生态系统应用的常见技术特性，运行在一般的商用硬件上，支持水平扩展,高可用。

官网：https://kudu.apache.org/

```
A new addition to the open source Apache Hadoop ecosystem, Apache Kudu completes Hadoop's storage layer to enable fast analytics on fast data.
```

Kudu 运行在一般的商用硬件上，支持水平扩展和高可用，集 HDFS 的顺序读和 HBase 的随机读于一身， 同时具备高性能的随机写，以及很强大的可用性（单行事务，一致性协议），支持与 Impala/Spark 计算引擎。

简单来说：Kudu 是一个与 HBase 类似的列式存储分布式数据库。
官方给 Kudu 的定位是：在更新更及时的基础上实现更快的数据分析

近年来 Kudu 的应用越来越广泛，在阿里、小米、网易等公司的大数据架构中，Kudu 都有着不可替代的地位。

3. Kudu的使用场景

- Strong performance for both scan and random access to help customers simplify complex hybrid architectures（适用于那些既有随机访问，也有批量数据扫描的复合场景）
- High CPU efficiency in order to maximize the return on investment that our customers are making in modern processors（高计算量的场景）
- High IO efficiency in order to leverage modern persistent storage（使用了高性能的存储设备，包括使用更多的内存）
- The ability to update data in place, to avoid extraneous processing and data movement（支持数据更新，避免数据反复迁移）
- The ability to support active-active replicated clusters that span multiple data centers in geographically distant locations（支持跨地域的实时数据备份和查询）

Kudu 的特性决定了它适合以下场景：

```
1、适用于那些既有随机访问，也有批量数据扫描的复合场景
2、CPU 密集型的场景
3、使用了高性能的存储设备，包括使用更多的内存
4、要求支持数据更新，避免数据反复迁移的场景
5、支持跨地域的实时数据备份和查询
```

从更实际角度来讲，我们不一定非要按照上面的场景来靠，如果未来可能会出现上述1的场景，当下的问题又能通过 Kudu 来解决，那么 Kudu 就是个比较好的选择。例如下面几个更实际的应用场景：

1、实时更新的数据应用(例如物联网数据产品)

> 刚刚到达的数据就马上要被终端用户使用访问到，未来还要做大规模的数据分析。

2、时间序列相关的应用(例如APM)。需要同时支持：

> 1、海量历史数据查询(数据顺序扫描)。
> 2、必须非常快地返回关于单个实体的细粒度查询(随机读)。

3、实时预测模型的应用(机器学习)

> 支持根据所有历史数据周期地更新模型。

3.1. Kudu 使用时的优势

（1）一个 table 由多个 tablet 组成，对分区查看、扩容和数据高可用支持非常好

（2）支持 update 和 upsert 操作。

（3）与 Imapla 集成或 spark 集成后（dataframe）可通过标准的 SQL 操作，使用起来很方便

（4）可与 spark 系统集成

3.2. Kudu 使用时的劣势

（1）只有主键可以设置 range 分区，且只能由一个主键，也就是一个表只能有一个字段 range 分区，且该字段必须是主键。

（2）如果是 pyspark 连接 kudu，则不能对 kudu 进行额外的操作；而 scala 的 spark 可以调用 kudu 本身的库，支持 kudu 的各种语法。

（3）kudu 的 shell 客户端不提供表 schema 查看。如果你不通过 imapla 连接 kudu，且想要查看表的元数据信息，需要用 spark 加载数据为 dataframe，通过查看 dataframe 的 schema 查看表的元数据信息。

（3）kudu 的 shell 客户端不提供表内容查看。如果你想要表的据信息，要么自己写脚本，要么通过 spark、imapla 查看。

（4）如果使用 range 分区需要手动添加分区。假设 id 为分区字段，需要手动设置第一个分区为 1-30，第二个分区为 30-60 等等

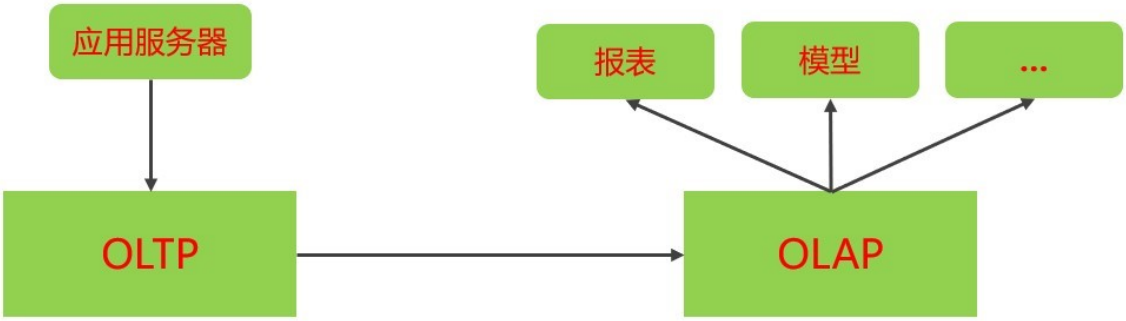
（5）时间格式是 utc 类型，需要将时间戳转化为 utc 类型，注意 8 个小时时差

4. Kudu 与其他存储对比

为了更好的对比，这里普及几个概念。

4.1. 相关概念

4.1.1. OLAP 和 OLTP



- 联机事务处理
- 面向业务开发人员
- 规范化的
- 为单条记录的查询做优化
- 行式存储

注意：OLTP、OLAP跟关系型数据库(RDBMS)没有直接关系

- 联机分析处理
- 面向分析师
- 非规范化的
- 星型/雪花型schema
- 为大规模分析查询做优化
- 列式存储

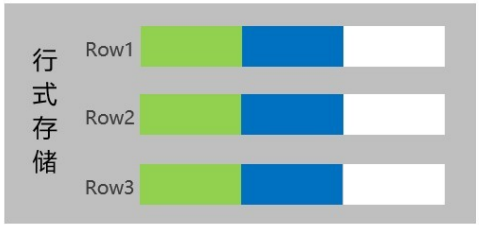
4.1.2. 行式存储和列式存储

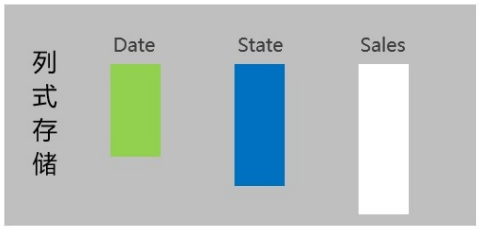
列式存储更适合OLAP场景

Table Sales

	Date	State	Sales
Row1	2019-01-01	MN	500
Row2	2019-01-01	CA	2,000
Row3	2019-01-02	CA	2,000

select date , sum(sales) from sales group by date;





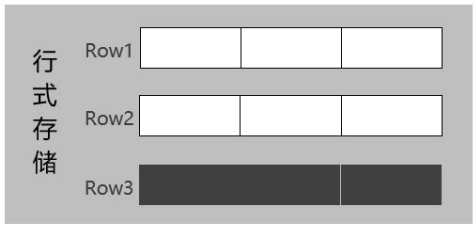
一般来说 OLTP 数据库使用行式存储，OLAP 数据库使用列式存储：

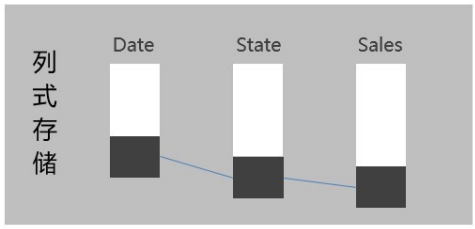
行式存储更适合OLTP场景

Table Sales

	Date	State	Sales
Row1	2019-01-01	MN	500
Row2	2019-01-01	CA	2,000
Row3	2019-01-02	CA	2,000

select * from sales where date=2019-01-02;





在查询某些列的数据时，行存储需要把整行数据读出来再过滤。

列存储不同列的数据分开存储在不同文件，面对分析查询(只涉及部分列)可以大幅降低 IO，因此列式存储相对于行式存储在 OLAP 场景下的优势可以这么来理解：

> 1、对于分析查询，一般只需要用到少量列，在列式存储中，只需要读取所需数据列即可。例如，如果您需要100列中的5列，则I/O减少20倍。
> 2、按列分开存储，按数据包读取时因此更易于压缩。列中的数据具有相同特征也更易于压缩，这样可以进一步减少I/O量。
> 3、由于减少了I/O，因此更多数据可以容纳在系统缓存中，进一步提高分析性能。

4.2. Kudu 和 RDBMS 对比

Kudu 跟关系型数据库有什么不同呢：

存储	存储结构	数据库类型	主键	非主键索引	外键	事务	SQL
MySQL/Oracle	行式存储	OLTP	有唯一主键	支持	支持	支持	支持
Kudu	列式存储	HTAP	有唯一主键	在路上	在路上	在路上，缺少对跨行原子性、一致性、隔离性、持久性事务的支持	不支持，需要跟Impala等SQL查询引擎搭配

怎么理解Kudu是HTAP（混合事务/分析处理，兼具OLTP和OLAP的一些特性)

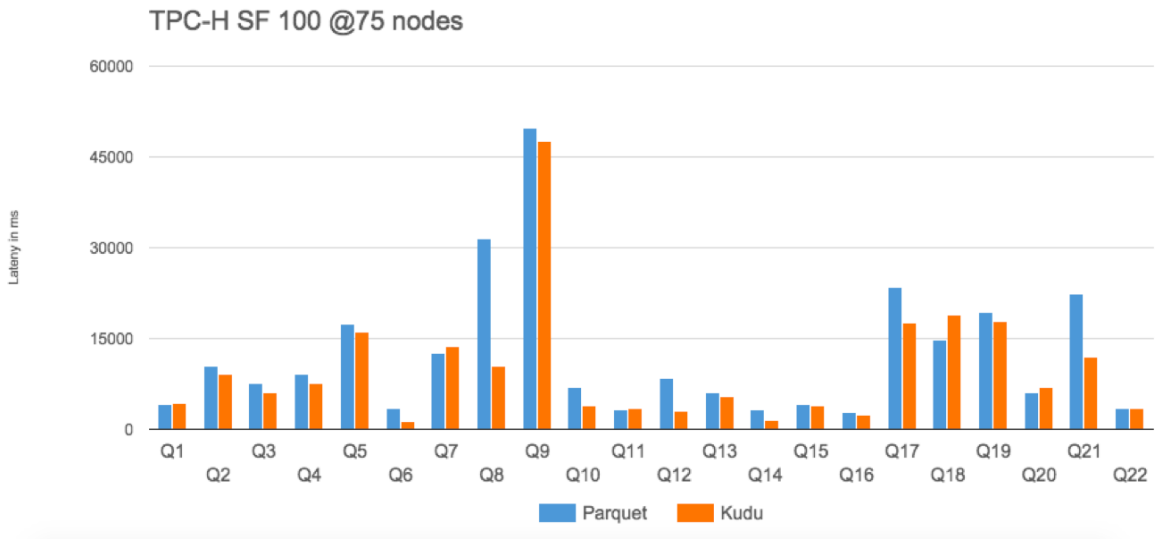
- 像OLAP：Kudu很容易跟Hive、Impala、SparkSQL等OLAP SQL查询引擎整合
- 像OLTP：Kudu支持根据主键快速检索，并且能够在对表进行分析查询时连续不断写入数据

4.3. Kudu 和 常见大数据存储对比

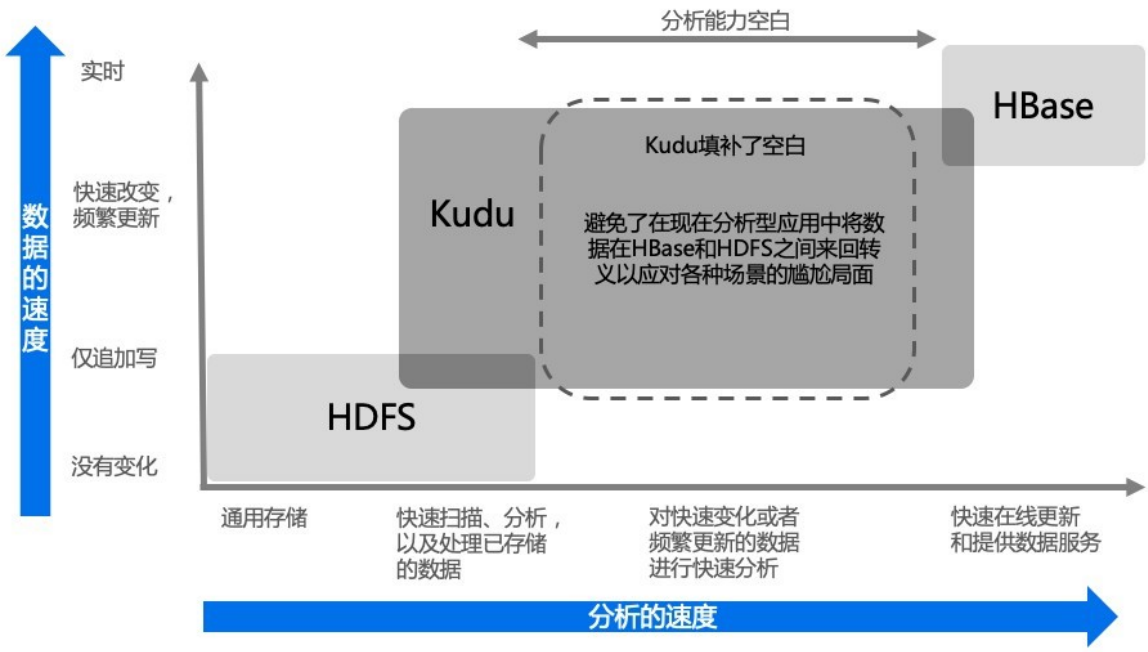
回归大数据存储领域，我们把 Kudu 跟 HDFS 和 HBase/Cassandra 做一个简单比较：

存储	大规数据扫描(分析场景)	随机读写
HDFS	非常擅长"全表扫描"（大规模数据分析）	不擅长随机读，不支持随机写（蹩脚实现：合并操作）
HBase/Cassandra	可以支持，但性能比 HDFS 差很多	擅长随机读写，按列族存储
Kudu	比 HBase/Cassandra 更擅长大规模数据扫描，接近HDFS上Parquet的性能	擅长随机读写，真正的列式存储，随机读写速度跟 HBase/Cassandra 接近

这里有一个 Kudu 跟 Parquet 的对比：



4.4. 对比总结



Kudu 不适合所有场景，他也不会完全取代 HDFS、HBase 等大数据存储组件。在有些场景下甚至 HDFS 或者 HBase 更为合适。

如果你正在找寻一款既支持实时查询又支持大规模数据分析和机器学习场景的存储，Kudu 将是一 个很好的选择，它将在搞定需求的同时极大降低运维复杂度。